Resource Allocation in Contending Virtualized Environments through Stochastic Virtual Machine Performance Modeling and Feedback

CONG-FENG JIANG, JIAN WAN, XIANG-HUA XU, JI-LIN ZHANG AND XIN-DONG YOU
Grid and Services Computing Technology Lab
Hangzhou Dianzi University
Hangzhou, 310037 P.R. China

In virtualized systems, allocation and scheduling of resources shared among multiple virtual machines faces challenges such as autonomy, isolation and high workload dynamics. The multiplexing and consolidation nature of virtualized systems also raise issues such as interference and conflicts among various virtual machine instances. Therefore traditional resource allocation strategy can’t achieve good performance without modifications according to these particular characteristics in virtualized systems. In this paper we use a stochastic model to characterize the resources (especially CPU) and workload dynamics. Then we use a weighted priority based service differentiation strategy to allocate resources in contending conditions to provide performance guarantees as well as load balance and fairness. In the proposed algorithms user behavior and workloads are characterized through the historical and real time performance profiling and estimation from hosted agents within individual Virtual Machines. The resources are allocated according to the demand as well as the performance of the targeted Virtual Machines based on the Sufferage aggregation and performance feedback. Experiments on a real Xen based virtualization environment with 20 Virtual Machines are conducted and evaluated for accuracy, efficiency, sensitivity, and overhead. The results show that the performance feedback based allocation can achieve a higher SLA satisfaction rate as 97.1%, a lower load imbalance index as 18.7%. The performance feedback based allocator uses 14.06% less CPU time for CPU-intensive applications and reduces 45.59% I/O wait time in disk contention environments. The results also show that the feedback based algorithm is valid, effective and scalable for implementation in real virtualized environments.

Keywords: resource allocation, virtualized environment, performance feedback, scheduling, workload characterization

1. INTRODUCTION

Virtualized systems are widely deployed in various environments including internet data centers and cloud computing environments for higher resource utilization, lower operational costs and less power consumption. Although in virtualized environments virtual activities are limited within a Virtual Machine (VM) and it seems that they are running independently and have no direct influence on real hardware, virtual activities can still affect real resource utilizations mutually when they are eventually mapping to real activities operated on the physical hardware. Therefore, it is nontrivial to implement...
fine-grained VM-level resource allocation into the Hypervisor or Virtual Machines Monitor (VMM) because fine-grained resource allocation and scheduling among multiple VMs can significantly reduce or coordinate request conflicts among VMs where the upper services and applications have contending or even conflicting resource demands.

In virtualized systems VMs are managed by the VMM and thus it is easier for VM migration for purpose such as load balance, energy efficiency, and fault tolerance. Therefore, fine-grained VM-level resource allocation can be easily realized in the VMM and thus can significantly reduce request conflicts among VMs where the upper services and applications have contending or even conflicting resource demands. However, the distributed nature of multi-layered virtualized environments makes traditional resource allocation approaches insufficient and induces new challenges including workload estimation and characterization, coordination of resource contention among multiple VMs, and heterogeneity in hardware capabilities. All these challenges make it impossible to apply traditional resource allocation approaches directly to the virtualized environments without modifications or adaptations because traditional approaches to resource allocation and scheduling are based on the fact that operating systems have full knowledge of and full control over the underlying hardware resources.

Moreover, resource contending is intensified in virtualized environments due to the consolidation nature of virtualization. In highly contended virtualized systems, resources should be allocated properly to VMs that can not only meet the individual VM’s performance requirements but also satisfy the overall requirements of the hosting system. The allocator should tradeoff between multiple VMs and multiple objectives such as overall performance and individual VM’s requirements in the perspective of service provider. Therefore, coordination and co-allocation of contending resources while providing performance assurance in a fair or prioritized manner to concurrent VMs and services is challenging, since they may have different Service Level Agreements (SLAs) and performance constraints. This requires the real time knowledge of workloads and performance feedback of the running services in order to provide an effective resource allocation decision. Due to the high density of services consolidation and the increasing number of users with heterogeneous requests, providing users with performance guarantees has become a crucial problem that needs to be addressed.

In order to achieve good resource allocation performance, the allocator requires the real time knowledge of workloads and application performance of the running services. Workload should be characterized prior to the allocation decision. However, in virtualized environments, the applications’ patterns can’t be profiled as easily as in real non-virtualized environments due to the sandboxing methodology of VMs. It’s difficult for the allocator to estimate the behavior of individual application running in a VM. Thanks to the managed nature of virtualization, the VMM can get the execution information of the VM and estimate the request patterns from the VM. Therefore, it is possible to provide enough information for resource allocation by integrating the knowledge of VMM and VM into the allocator.

In this paper resource allocation algorithms for contending environments are proposed to minimize performance losses in virtualized system with various constraints. We give an example in Fig. 1 to demonstrate the essential of resource co-allocation and optimized scheduling in virtualized environments. Fig. 1 shows that the CPU and disk operation varies over time considerably, and the peaks of the two resource types occurred
at different times of the day. This suggests that static resource allocation and over-provisioning can meet application SLAs only when the resources are allocated for peak demands and thus results in resource wastes. Specifically, in Fig. 1, enough resources are allocated for various VMs at system startup and fixed after the initial allocation. However, in our allocator, only sufficient resources are allocated to VMs at system startup and regulated dynamically after the initial allocation. The results show that our allocator can not only provide better performance guarantees but also smooth the resource accesses and utilizations by redistributing and heavily overlapping VM workloads with consideration of mutual dependencies among multiple VMs.

In this paper, we argue that the ultimate goal of resource allocation in contending virtualized environments is to provide predefined desirable SLAs and performance guarantees with minimal resource utilization including CPU cycles, memory capacities, disk or network bandwidth, and energy. In other words, the ultimate goal of resource allocation in virtualized environments is to achieve maximal or best performance under predefined resource budget and constraints.

To achieve this goal, firstly, the performance model and parameters of a virtualized system should be identified to characterize the relationship between the resource allocation and system performance and to predict application behavior with high accuracy. Secondly, based on this performance model, appropriate resource allocation decisions should be made and executed in certain manner to control these performance parameters in an optimal range. This also leads to the problem that how to develop an allocation or scheduling model based on the deduced parameters. Thirdly, the real performance of the targeted system should act as a persistent feedback source to the resource allocator to justify, correct, and improve the control accuracy, effectiveness, and scalability for the future incoming dynamic workloads. We proposed an allocation framework to address the allocation problem which is illustrated in Fig. 2.
implement the performance monitoring, training, and resource allocation. We also compare the proposed resource allocation model with existing resource allocation models and solutions. Real workload experiment results show that optimized performance could be achieved, if we integrate the performance model and Sufferage based prioritization mechanism into resource allocation and scheduling algorithms in contending virtualized systems.

In summary, this paper has made fundamental contributions in four aspects:

(1) Design and evaluation of a new stochastic performance model including resource states presentation and performance transition. The state based model is more scalable than constant allocation strategy when runtime conditions change dynamically and frequently. The investigation of combining multiple types of system parameters into the resource allocation problem, such as resource failure probabilities and workload uncertainty, will extend the applicability of this resource allocation strategy into real problem domains where these random conditions occur simultaneously.

(2) Design and evaluation of new resource allocation models in contending virtualized environments with SLA requirements. We introduce Sufferage-based prioritization into contending resource allocation. The SLA utility can be maximized when the performance budget is constrained and vice versa. The proposed models can be applied to provide different assurance for a specific objective, such as power efficiency, workload balancing, etc.

(3) Design and evaluation of the workload monitoring, characterization and actuation agent in virtualized systems. The agent consists of adaptive application performance monitoring, prediction and behavior identification functions, and can be applied to enable efficient and distributed resource allocation in existing resource allocation heuristics according to the real time profiling based on the historical data. The implementation framework of the proposed agent is proven effective and highly scalable on real workload in contending virtualized environment.

(4) After extensive performance study of some resource allocation algorithms, we reveal their relative strengths, weaknesses, and applicability in scalable contending virtualized environments. In particular, the experiment results suggest that it is more accurate to integrate hardware and software bi-level profiling for application behavior identification and performance prediction, instead of using them separately in virtualized systems.
The remainder of this paper is organized as follows: In section 2, we define a stochastic performance model and a resource allocation agent for the resource allocation problem for multiple VMs with static and dynamic workloads. Then in section 3, resource allocation algorithms including Sufferage-priority based service differentiations are proposed, which account for SLA insurance, performance, fairness, and resource utilization. In section 4 we discuss the experimental testbed in a Xen based virtualized systems and evaluate the algorithms in experiments, and we make extensive performance analysis. We discuss related work in section 5. Finally we conclude our contribution in section 6 and discuss with some directions for future work.

2. STOCHASTIC PERFORMANCE MODEL OF VIRTUALIZED ENVIRONMENTS

In the context of resource allocation, resource may represent any resource that affects the application metrics of interest and that can be allocated between the jobs or tasks. However, for simplicity, we use CPU, memory and disk I/O as the resources in the scope of this paper. To achieve the automation, adaptation, and scalability of resource allocation, the first problem we must address is characterizing the relationship between resource allocation and application performance because we cannot do anything unless we have a good understanding of the performance and the resources allocated. Moreover, it is more important to correctly model application performance and resource allocation in contending virtualized systems, where multiple applications share the same physical resources. In virtualization environments, incorrect resource allocation decision to a specific VM can lead to the wrong reactions not only of the targeted VM but also any or all of the other VMs running on the same physical platform.

To model the performance and resource allocation of virtualized system, we consider a targeted virtualized system consisting of $M$ physical machines (i.e., the service source) each with an associated queue to which incoming workloads including migrations may be directed. In this system, each physical machine has $Q$ kind of resources (here $Q = 3$, i.e., resources are CPU, memory and disk I/O) to be allocated when available. We assume that each physical machine contains $N$ VMs (excluding the Dom0 instance) and each VM have $P$ applications running within it. Each VM has its CPU time cycles, memory capacities, disk I/O bandwidths. For generality, we assume that each application can request any kind of aforementioned resources separately or simultaneously. Other assumptions are made in the following paragraphs. For the convenience and simplicity of presentation, we may use workload, job, and task interchangeably to stand for the resource requests. We may also use allocator or scheduler interchangeably to refer to the manager who is responsible for resource allocation.

To focus on the resource allocation problem, we omit the details of how to estimate the execution time of tasks and available time of resources. The assumption that estimation of expected task execution times on each machine and the real-time states of the resources are known is commonly used in scheduling algorithms and it is well studied and implemented. Approaches and techniques such as statistical prediction for doing this estimation can be found in various literatures [28-30]. The general model under consideration in this paper can be described in the following sections.
2.1 Stochastic States and Transitions

In this paper, the relationship between the system performance and the resource allocation is modeled as random variable and we assume that stochastic information can be obtained for characterizing the probability [1]. In a real virtualized system, we use the historical information regarding past performance and resource allocation for a given workload to approximate the probability of variations of system performance parameters with respect to the resource allocation. Here, we consider the resource models with 3 distinct operational states, i.e., idle and ready for services (state 1), busy but available for services in the future (state 2), not available for services (state 3). The state transition graph is illustrated in Fig. 3. Please refer to [1] for model details.

![Fig. 3. The state transition graph of resources; The transition probability is also illustrated in the graph.](image)

The transition probability matrix of resources can be summarized as Eq. (1).

\[
P = \begin{bmatrix}
0 & p_1 & q_1 \\
q_2 & 0 & p_2 \\
p_3 & q_3 & 0
\end{bmatrix}
\]

Thus, the steady-state probability of a given resource being in a particular operational state can be calculated. Here, the transition probability of resources after \(n\) steps from state 1 to each state for its first time can be calculated by discretization of observed data as follows [1]:

\[
f_{1i}^{(n)} = \begin{cases}
(q_1p_3)^{n-1}q_3q_1, n = 2m, m \geq 1 \\
(q_1p_3)^n p_3, n = 2m + 1, m \geq 0
\end{cases}
\] (2)

\[
f_{1i}^{(n)} = \begin{cases}
(p_2q_2)^{n-1}q_1p_3, n = 2m, m \geq 1 \\
(p_2q_2)^n q_1, n = 2m + 1, m \geq 0
\end{cases}
\] (3)

\[
f_{1i}^{(n)} = \begin{cases}
0, n = 1 \\
p_3(p_2q_2)^{n-1}q_3 + q_1(p_2q_2)^{n-1}p_3, n = 2m, m \geq 1 \\
p_3(p_2q_2)^n p_3 + q_1(p_2q_2)^n q_3, n = 2m + 1, m \geq 1
\end{cases}
\] (4)

Similarly, the workloads and applications also have 4 distinct states, i.e., waiting
(state 1), running (state 2), completed (state 3), and failed (state 4). In state 1, workloads arrive to a VM in a Poisson stream but there is no resource available for service. In state 2, workloads in waiting queue are served by the available resources with some service times. In state 3, the workloads have been completed successfully. In state 4, the workloads are failed due to hardware failures, system crashes, or some else. Thus, the state transition graph of workload (or applications) is illustrated in Fig. 4.

Fig. 4. The state transition graph of workloads; The transition probability is also illustrated in the graph.

The transition probability matrix of resources can be summarized as follows:

\[
P = \begin{bmatrix}
0 & s_1 & 0 & r_1 \\
r_1 & 0 & s_2 & r_2 \\
0 & 0 & 0 & 0 \\
s_3 & 0 & 0 & 0
\end{bmatrix}
\]  
(5)

Therefore, the transition probability of workloads after \( n \) steps from state 1 to each state for its first time can be calculated by discretization of observed data as follows [1]:

\[
f^{(n)}_{12} = (r_1s_1)^n s_1, n = 2m + 1, m \geq 0
\]
(6)

\[
f^{(n)}_{13} = \begin{cases} (r_1s_1)^n s_1s_2, n = 2m, m \geq 0 \\ (s_1r_1)^y(s_1r_1)^y, n = 2m + 1, x + y = m, m \geq 0 \end{cases}
\]
(7)

\[
f^{(n)}_{14} = \begin{cases} (s_1r_1)^{n-1}s_1r_2, n = 2m, m \geq 1 \\ (s_1r_1)^y r_1, n = 2m + 1, m \geq 0 \end{cases}
\]
(8)

\[
f^{(n)}_{11} = \begin{cases} 0, n = 1 \\ s_1r_1 + s_1s_1, n = 2 \\ s_1r_2s_1, n = 3 \end{cases}
\]
(9)

Here note that in our resource allocation scenario, we do not consider the transition probability from the same state in one step since it means the resource or workload state does not change.

To model the state of the resources and virtual machines, a simple approach is to count the corresponding instances in the data and to measure its behavior under load from a workload generator. We use different virtual machine configurations and work-
load parameters to obtain a statistically result for the virtualized environment. The data collection and processing functions are implemented in the virtual machine monitor so that it is independent of the guest virtual machines and underlying specific physical resources like processors, memories, and hard disks. After the benchmarking, we can obtain the results with a target threshold of confidence and accuracy.

2.2 Application Performance Modeling

We use an extended conditional performance model [2] to characterize the relationship between the workloads, resources and the application performance. Based on the stochastic model described in the previous section, we use a multi-dimensional state space to model this relationship. Specifically, we use the discretized data from extensive experiments to distinguish the dependencies and effects of resource allocation on application performance and to find an accurate function to approximate it. For simplicity, we construct a distribution of response time conditioned on random variables including work-load states, resource states and resource allocations. For the convenience of formulation, we summarize the notations that we use in this paper.

<table>
<thead>
<tr>
<th>Table 1. Notations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
</tr>
<tr>
<td>$C, c$</td>
</tr>
<tr>
<td>$M, m$</td>
</tr>
<tr>
<td>$D, d$</td>
</tr>
<tr>
<td>$W, w$</td>
</tr>
<tr>
<td>$R, r$</td>
</tr>
<tr>
<td>$\tau, t$</td>
</tr>
<tr>
<td>$T_t$</td>
</tr>
</tbody>
</table>

Thus, we can derive the conditional probability of application response time conditioned on workload characteristics and resource allocation, i.e., $p[t | w, r]$, as shown in the following:

$$p[t | w, r] = \sum_{m \in M} \sum_{c \in C} \sum_{d \in D} p[t | r, w, c, m, d] \cdot p(c, m, d | w, r)$$  \hspace{1cm} (10)

$$P[\tau \leq T_t | w, r] = \sum_{c \in C} \sum_{m \in M} \sum_{d \in D} p[x | r, w, c, m, d] \cdot p(c, m, d | w, r)$$  \hspace{1cm} (11)

Linear regression [3] is a simple regression method that attempts to model the relationship between variables by fitting a linear equation to observed data. Here we use least squares regression to aggregate the probability in our performance model. To evaluate the accuracy of our model, we use the median, average, standard deviation, and 90th percentile values for prediction error for validation.

For example, in order to compute the response-time under different workloads and server configurations, we use a simple model as:
\[ R = \frac{1}{(a - b \times \lambda)}. \]  

Where \( R \) is the response time, \( \lambda \) is the workload, and \( a \) and \( b \) are constants that depend on the settings and can be computed under different tuples of \((\lambda, R)\). Once the model is constructed, we can predict the response time under the next workload. The model is periodically updated and iterated by using the new \((\lambda, R)\) tuples during prediction. In order to get an accurate model, the available tuples should cover the whole problem space. Please note that although the polynomial model of \((\lambda, R)\) can get a higher accuracy, it is unstable and introduces oscillation and is not guaranteed to converge.

For simplicity and less overheads, the resource utilization under workload is calculated as follows:

\[ U(\lambda) = k\lambda + C. \]  

Where \( U(\lambda) \) is the utilization, \( \lambda \) is the workload, and \( k \) and \( C \) are constants that depend on the settings and can be computed under different tuples of \((\lambda, U)\). \( C \) is the resource utilization when the virtual machine is idle. The computation of the utilization model is the same with the response time model. Since multiple virtual machines share the same physical resources, shared resources may become the bottleneck of the whole applications, especially for critical conditions. For example, the performance degrades significantly when the number of virtual machines increases.

Assume that \( U(\lambda)_{\text{max}} \) is the maximum utilization (for example, 0.9 for memory utilization) and \( \lambda_c \) is the contributed workload, then

\[ U(\lambda)_{\text{max}} = k\lambda_c + C. \]  

To train the model, we use some test instances. For example, \( k \) can be computed for available \( P_1 \)

\[ k = \frac{U(P) - C}{P}. \]  

Based on Eqs. (14) and (15), we can compute the workload contributing the maximum utilization and performance bottleneck:

\[ \lambda_c = \frac{U_{\text{max}} - C}{k} = \frac{U_{\text{max}} - C}{U(\lambda) - C} = \frac{\lambda \times (U_{\text{max}} - C)}{U(\lambda) - C}. \]  

However, since the relationship between workload and performance is near linear when there is no bottleneck in resources, we should recalibrate the model, i.e., the value of \( k \) and \( C \). Suppose for time interval \( t \), the value \( s \) of \( k \) and \( C \) are \( k_t \) and \( C_t \). For time interval \( t + 1 \), the value \( s \) of \( k \) and \( C \) are \( k_{t+1} \) and \( C_{t+1} \). Then we can get:

\[ k_t = \frac{U(\lambda_{t+1}) - U(\lambda_t)}{\lambda_{t+1} - \lambda_t} = \frac{\Delta U}{\Delta \lambda}. \]
\[ C_i = U(\lambda_i) - k_i \lambda_i = U(\lambda_i) - \lambda_i \times \frac{\Delta U}{\Delta \lambda}, \]  

(18)

\[ \lambda_i = \frac{U_{\text{max}} - \left( U(\lambda_i) - \lambda_i \times \frac{\Delta U}{\Delta \lambda} \right)}{\frac{\Delta U}{\Delta \lambda}} = \frac{U_{\text{max}} - U(\lambda_i)}{\frac{\Delta U}{\Delta \lambda}} + \lambda_i. \]  

(19)

Eq. (19) suggests that the workload converges to the critical value \( \lambda_c \) in monotone increasing utilization.

### 2.3 Workload Characterization and Estimation

In this paper, we argue that based on an accurate workload characterization model, we can provide ability to predict a virtualized application’s performance and to implement near-optimal resource allocation in virtualized data centers based on the ability at a given allocation level of partitionable resources and an observed competition level of non-partitionable resources. In our approach, the virtual machine monitor is responsible for allocating basic resources such as CPU slices, memory capacities, and disk and network I/O bandwidth. In runtime, resource allocation is automated based on the workload characterization and the stochastic model described in the above subsection. Once the allocation is made, a specific share of physical resources to a VM will be allocated automatically and the resulted specific performance can be measured using application-specific performance metrics such as response time and/or throughput, to aid for the next cycle of allocation.

We use statistical sampling and thread-level estimation for workload characterization, including behavioral quantification, performance estimation, measurement, and monitoring. In our practical implementation, this is done via an adaptive agent inside a VM. The agent is responsible for collection and analysis of workload traces, performance data, and real time resource utilizations. It also provides interfaces for repeatable and verifiable experiments and functions to allow detailed and flexible emulation of enterprise-class workloads. Fig. 5 illustrates our agent based workload characterization and behavior identification framework.

More specifically, the model estimator consists of the following steps:

**Step 1:** Data aggregation. The estimator aggregates the data such as resource consumption, resource allocation, and application performance data from the traces of past performance and past allocation, expected performance, and real time profiling.

**Step 2:** Data preprocessing. The aggregated data are preprocessed before decision making. For example, the data trace will be synchronized and smoothen for usage in the estimation process.

**Step 3:** Data partitioning. We partition the aggregated data into two kinds, one is training data and the other is testing data.
Step 4: Estimation. We use the training data to generate a proper model to characterize the relationship between the resource allocation and the performance of the targeted virtual machines. For simplicity and efficiency, we only use linear regression model for estimation. Here note that the model estimator is module-based and the implementation can be easily replaced by different approaches. For an instance, we can use different approaches trade-offing higher calculation cost and required hardware resources.

Step 5: Model evaluation. We use the second kind of the aggregated data, i.e., the real trace data to evaluate our estimation and allocation decisions. The variation between our calculated estimation and the real performance is also calculated and referred to modify and improve our estimation model.

Since application performance and resource consumption may change over time, the profiling module of the nested agent must be updated online as new sets of observations are made, reflecting the changes in the system state and application behavior. Moreover, if VMs are migrated from one physical host to another, the agent will also be running in the targeted host. Here note that we neglect the hardware heterogeneity of the targeted migrated host and assume that a new agent could be constructed from scratch on the targeted host following the same principles as used to construct the original agent. This can be done through an abstraction level and programming techniques. For example, we can use the Java Native Interface (JNI) to avoid the processor architecture and operating system design variance and therefore the agent model is generically applicable in a variety of real-world virtualization environments.

For simplicity and less resource consumption, we use linear regression model for estimation. However, since the estimator is module based, it can be easily replaced by other approaches.

3. RESOURCE ALLOCATION WITH QOS GUARANTEES AND SERVICE DIFFERENTIATIONS

In virtualized system, multiple applications are consolidated to share dynamically allocated resources to reduce infrastructure and operating costs while simultaneously increasing resource utilization. Thus, it must be able to undertake complex management
over various resources such as provisioning, deployment, execution and adaptation in an
autonomic way. This requires the resource allocator to be efficient, accurate, lower over-
head and at the same time fulfilling the SLAs agreed with the guest VMs. As a result, the
system administrators are faced with growing challenges to meet SLAs in the presence of
dynamic resource sharing, conflicting, and unpredictable interactions across many VMs.
In order to provide SLAs, jobs may be differentiated through prioritized, i.e., a job with
higher index is processed at each time, which in this case depends on the cumulative
amount of processing time. In homogeneous machines, the optimality of priority-index
policies has been extended to restricted classes of models for resource allocation. Under
this situation, the main performance objectives include the minimization of the applica-
tion response time, and minimization of expected finishing time of the last workload. In a
heterogeneous virtualized computing system, the running times may differ depending on
which computer executes the given job. In this case, after migration to heterogeneous
machines, the workload characteristics may change too. Resource allocation in such het-
erogeneous computing environments has been proven NP-hard in general (e.g., [4]).
Various heuristics for resource allocation have been proposed for non-virtualized sys-
tems, e.g., [5-7]. In this paper, we consider the SLAs requirements as constraints of the
resource allocation problem and propose heuristics to solve it.

3.1 SLA Description and Validation

In our implementation, within each physical host, the VMM controls the allocation
of physical resources to individual VM. A change in the resource demand of various
VMs may require reconfiguring one or more VM allocations such that the system per-
fomance and SLA-based requirements can be guaranteed. It is also desirable to allocate
contending resources among various VMs while still satisfying their SLAs requirements
in response to changing conditions.

Let \( Q \) denote a set of SLA requirements, \( Q = \{ Q_i | i = 1, 2, ..., n \} \), \( Q_i = (T_i, R_i, S_i, A_i, P_i) \), where:

- \( T_i \) is timeliness requirement,
- \( R_i \) is reliability requirement,
- \( S_i \) is security requirement,
- \( A_i \) is accuracy requirement,
- \( P_i \) is priority requirement.

For simplicity, we use discrete values to modeling the SLA requirements, i.e., the requirements are presented by several
levels like very low, low, medium, high, and very high, not a specific number like 10% or 90% because in real cloud computing system with user interaction a user only cares the interactive experience, not the specific performance numbers.

3.2 Service Differentiation through Sufferage Priority

Here we define a contention index for representing the contending intensity of the
dedicated platform as follows:

\[
c_j = \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} p_i},
\]

(20)
where $c_i$ stands for the $i$th contention index, for example, contention index of CPU or memory;
$r_i$ stands for the expected requesting resources by individual VM;
$p_i$ stands for the total resources provided by the underlying machines.

For fairness and efficiency, in a non-contending environment, resources are allocated fairly as demanded. In contending cases, we use a priority mechanism to differentiate and coordinate the resource allocation among multiple VMs. In order to avoid the side-effects imposed by priority mechanism such as workload imbalance, prolonged response time, and violations of SLA requirements, the value of priority is not set deterministically and randomly. Instead, the priority value is proportional to sufferage [8] values of the specific VM, i.e.:

$$SufferageValue = (c_{before} - c_{after}) \times (1 + SLA_{value})$$ (21)

where $c_{before}$ stands for the contention index before the resource allocation is satisfied for the dedicated VM;
$c_{after}$ stands for the contention index after the resource allocation is satisfied for the dedicated VM;
$SLA_{value}$ stands for the SLA requirement of the specific VM, and SLA requirement is a normalized value.

Therefore, in contending conditions, the priority value of contending VM is computed according to its sufferage value. This kind of prioritization mechanism have two salient features: (1) it can avoid contention among multiple VMs, and (2) it can leverage the contention conditions significantly and efficiently when contention has already happened.

### 3.3 Resource Allocation and Scheduling

Not surprisingly, the resource allocation problem is NP-complete. For instance, considering two servers that provide a single resource and services that have needs for that resource, the resource allocation trivially reduces to 2-partition, which is known to be NP-complete [9]. Due to space limitations, we refer the reader to a technical report for the straightforward proof [10]. Here we use heuristic algorithms to solve this problem of resource allocation with SLA constraints. The algorithms are triggered periodically and increase or decrease the resource allocated to a dedicated VM dynamically. The time-varying workload parameters, such as workload intensity, real-time resource utilization, are specified as the allocating variables that are used to parameterize the allocation model. The utilization of allocation parameters can be generalized to accommodate more sophisticated workload characterizations and more complicated multiple server environments in real scenarios.

The proposed algorithms are listed as follows:

1. On demand allocation: In on demand allocation, the resource demand will be satisfied by the allocator if there are resources available to be allocated. This kind of allocation does not consider the real time performance of the targeted VM. However, we assume that the VM scheduler employs a capped mode, such that a VM cannot use
more than the maximum amount of resource allocated to the specific VM.

(2) Greedy allocation: In the greedy allocation, the resource demand will be satisfied by
the allocator if there are resource available to be allocated and the amount of resource
allocated can be as much as the total resources of the physical machine. Similarly,
this kind of allocation does not consider the real time performance of the targeted
VM either.

(3) Performance feedback based allocation: In the performance feedback based allocation,
the resource demand only be satisfied when the allocation can make the system per-
formance better, for example, to make the resource earlier available with higher tran-
sition probability, or to make the workload earlier finished with higher transition
probability. In other words, more resources will be allocated only when better per-
formance can be achieved and less resource will be allocated only when there are less
performance degradations after the allocation. The core concept of the feedback based
resource allocation can be listed as the following pseudo codes:

\[
\text{Feedback Based Amount adaptation for resource allocation}
\]

for every VM
  for every kind of resource
    if resource allocation is increasing
      if the performance is improving
        increase allocation amount
      else
        if the performance is not changing
          decrease allocation amount

Here we use a normalized cost-performance ratio, \( cpr \), to indicate the revenue
of resource allocation to dedicated VMs. \( cpr \) is defined in the following:

\[
cpr = \frac{\text{performance increment}}{\text{resource increment}}. \tag{22}
\]

In the above equation, resource increment and performance increment are percent
values. When resource allocation is increasing, higher \( cpr \) represents better gained per-
formance. When resource allocation is decreasing, higher \( cpr \) represents worse gained
performance.

It seems that there may be deadlocks in the resource allocation process. However, it
is the oscillation problem in resource allocation and the virtual machine performance.
Since all the physical resource allocation decisions are made via the virtual machine
monitor and the individual virtual machines are only resource consumers. The resource
allocation algorithms are triggered periodically if the utilization and/or performance are
breaking the predefined thresholds. It is more like a one-way manner. For example, if the
performance of targeted virtual machines does not change that much, it means the per-
formance is kept in the predefined domains. We use control theory to reduce the oscilla-
tion and smooth the performance curve in the feedback based approach.

Please note that the hosted agents can always characterize the user behavior and
workloads. However, the on-demand allocation and greedy allocation algorithms pro-
posed in the paper do not consider the real time performance of targeted VM, therefore they do not need the historical characterization information.

3.4 Load Balancing and VM Migration

Live migration of VMs is used for optimal CPU and memory allocation, load balancing, and improved overall utilization of hardware resources. In this paper, based on the workload characterization model, we use the resource allocation agent to recommend workload redistribution in order to mitigate hot-spotting or balancing. We use $wbi$ as the main metric for load balancing across multiple resources. $wbi$ is a normalized load metric across devices or platforms and it is between the minimum and maximum normalized values.

Let $w_{\text{max}}$ denote the maximum $wio$ (queue length of waiting I/O) of all VMs, and the minimum is $w_{\text{min}}$, we define a workload balance index as follows:

$$cpr = wbi = \frac{w_{\text{min}}}{w_{\text{max}}}.$$ (23)

$wbi$ can be any value in range $[0, 1]$. If $wbi$ equals 1, it suggests that workload is balanced among VMs. And if $wbi$ equals 0 it suggests that workload is imbalanced among VMs and at least there is one VM which is not allocated any workload. In the resource allocation process, two thresholds, $wbi_l$ and $wbi_h$, $wbi_l < wbi_h$ are selected. The allocator sets $wbi$ to 0 before allocating. Then workloads are processed until $wbi > wbi_h$ and then, queue aware scheduling is used until $wbi < wbi_l$. Appropriate $wbi_l$ and $wbi_h$ can be got after several control cycles.

In a loop, load will not be redistributed unless its $wbi$ is larger than the upper boundary. Similarly, a physical platform will not accept a migration unless its $wbi$ is less than the lower boundary and its $wbi$ is still lower than the upper boundary after migration onto it. All the migration is done pair-wise such that both platforms and devices approach the overall average of $wbi$. In each loop the allocator tries to find the proper set of VMs that need to be migrated from the host with maximum $wbi$ to the one with the minimum $wbi$.

4. EXPERIMENT RESULTS AND PERFORMANCE ANALYSIS

We use a Xen-based testbed to evaluate our approach and to model the application performance in a contending virtualized environment. In the experiment settings, the host consists of an I/O driver domain, i.e., Dom0 which routes the entire disk I/O from user domain, i.e., DomU. In our implementation, we use the default credit scheduler in all the VMs, including the dom0 VM. The CPU allocation generic parameter, such as the CAP parameter, is set into an initial upper bound and can be changed dynamically by our allocator from within Dom0 at run time. All the benchmarks are listed in Tables 2 and 3. The testing benchmarks and applications include RUBiS [31], TPC-W [32], httperf [33], Oracle and SQL Server2005 databases, an eMule bittorrent application, one customized ftp application and one program for searching prime numbers.
RUBiS is an online e-commerce test application with a backend MySQL database component, and a client program to generate workload. The application generates a lot of requests that are sequential or simultaneous and we use it to compare it with others with random access pattern. The benchmark records the number of users, minimum and maximum delay between transactions, response times, and percentage of each type of transactions, etc. To allocate CPU resources strictly, we use CPU pinning techniques to bind the targeted VMs to the specific virtual CPU (vCPU).

Table 2. VM configurations and benchmarks on server 1.

<table>
<thead>
<tr>
<th>ID</th>
<th>VM</th>
<th>Memory (MB)</th>
<th>vCPU</th>
<th>Benchmarks and applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dom0</td>
<td>2478</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>Linux01</td>
<td>1023</td>
<td>2</td>
<td>RUBiS</td>
</tr>
<tr>
<td>2</td>
<td>Linux02</td>
<td>1023</td>
<td>2</td>
<td>TPC-W, Oracle</td>
</tr>
<tr>
<td>3</td>
<td>Linux03</td>
<td>511</td>
<td>2</td>
<td>ftp, prime</td>
</tr>
<tr>
<td>4</td>
<td>WinXP</td>
<td>1031</td>
<td>2</td>
<td>SQL Server2005,eMule</td>
</tr>
</tbody>
</table>

Table 3. VM configurations and benchmarks on server 2.

<table>
<thead>
<tr>
<th>ID</th>
<th>VM</th>
<th>Memory (MB)</th>
<th>vCPU</th>
<th>Benchmark and applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Dom0</td>
<td>2478</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>Linux05</td>
<td>256</td>
<td>1</td>
<td>prime</td>
</tr>
<tr>
<td>6</td>
<td>Linux06</td>
<td>256</td>
<td>1</td>
<td>prime</td>
</tr>
<tr>
<td>7</td>
<td>Linux07</td>
<td>256</td>
<td>2</td>
<td>prime</td>
</tr>
<tr>
<td>8</td>
<td>Linux08</td>
<td>256</td>
<td>2</td>
<td>prime</td>
</tr>
<tr>
<td>9</td>
<td>Linux09</td>
<td>256</td>
<td>2</td>
<td>httpperf</td>
</tr>
<tr>
<td>10</td>
<td>Linux10</td>
<td>256</td>
<td>2</td>
<td>httpperf</td>
</tr>
<tr>
<td>11</td>
<td>Linux11</td>
<td>512</td>
<td>2</td>
<td>httpperf</td>
</tr>
<tr>
<td>12</td>
<td>Linux12</td>
<td>512</td>
<td>2</td>
<td>httpperf</td>
</tr>
<tr>
<td>13</td>
<td>Linux13</td>
<td>512</td>
<td>2</td>
<td>ftp+ httpperf</td>
</tr>
<tr>
<td>14</td>
<td>Linux14</td>
<td>512</td>
<td>2</td>
<td>ftp+ httpperf</td>
</tr>
<tr>
<td>15</td>
<td>Linux15</td>
<td>512</td>
<td>4</td>
<td>ftp+ httpperf</td>
</tr>
<tr>
<td>16</td>
<td>Linux16</td>
<td>512</td>
<td>4</td>
<td>ftp+ httpperf</td>
</tr>
<tr>
<td>17</td>
<td>WinXP17</td>
<td>512</td>
<td>2</td>
<td>prime+ httpperf</td>
</tr>
<tr>
<td>18</td>
<td>WinXP18</td>
<td>512</td>
<td>2</td>
<td>prime+ httpperf</td>
</tr>
<tr>
<td>19</td>
<td>WinXP19</td>
<td>1024</td>
<td>4</td>
<td>SQL Server2005,eMule</td>
</tr>
<tr>
<td>20</td>
<td>WinXP20</td>
<td>1024</td>
<td>4</td>
<td>SQL Server2005,eMule</td>
</tr>
</tbody>
</table>

TPC-W is run in a closed loop, where a small number of processes generate requests with zero think time between the completion of a request and issue of next request. Also the setup is a two tier environment, where load is generated on the first tier (client) and submitted to the back end database. In this paper, we used a workload mix called the browsing mix that simulates a user browsing through an auction site.
Httpperf is a tool to measure web server performance via HTTP protocol. It offers a variety of workload generators to generate a fixed number of HTTP GET requests and to measure how many replies (responses) came back from the server and at what rate the responses arrived.

We use an ftp server to represent the disk I/O contention among various VMs. The ftp server runs a certain number of concurrent threads, each serving a client that continuously downloads files from the server. An ftp client can request an encrypted or unencrypted stream and in different protocols. The server reads the particular file from the disk and sends it to the client when it receives a download request. In this case, reading a file from the disk consumes disk I/O resource. For a given number of threads, by changing the fraction of the client requests for encrypted data, we can vary the amount of CPU or disk I/O resource used. This flexibility allowed us to study our scheduler’s behavior for CPU and disk I/O bottlenecks.

These representative benchmark set includes CPU-intensive, disk I/O intensive, network I/O intensive, and the combination of them and they are running with varied resource configurations of a real virtualized system. These applications contains CPU, memory and disk I/O requests which can be split into various components such as data access, index access, log writing, etc. To understand the effect of contending devices in virtualized environment, all the workload is running in isolated and simultaneous way on the underlying devices.

We capture the parameters including CPU utilization, memory utilization, and disk I/O waiting size as main representatives of the real time system performance and conditions. Our agents periodically collect two types of statistics: real-time resource utilizations and performance of applications. For example, CPU utilization statistics are collected using Xen’s xm command while disk utilization statistics are collected using the iostat command, which is part of the sysstat package of the system. In our implementation, we used a timer-like script to measure both the application throughput and the server-side response time directly from the application, where throughput is defined as the total number of client requests serviced, and for each client request, response time is defined as the amount of time taken to service the request. In a real system, application-level performance may be obtained from application logs or from dedicated tools.

The dynamic memory allocation is made by the `xm mem-set` command in Dom0 mode. In our experiments, we use the dom0 blkback/netback process running inside Dom0 for memory priority allocation using an ionice values. The xentop tool is used to collect the I/O requests statistics of various VMs, such as reads, writes, cache misses, etc. In order to avoid arbitrary magnitude, we use normalized performance rather than the absolute performance. All the experiment results are listed in the followings figures.

In our experiments, the performance feedback algorithm outperforms the on-demand and greedy allocation algorithms in most of the performance metrics, such as CPU utilization, memory utilization, disk I/O utilization and application response time. Therefore, we only provide the results of the performance feedback based allocator.

4.1 Accuracy

The first goal of our allocator is to detect and mitigate resource bottlenecks in multiple resources and across multiple application tiers. And from the results we can see that
for different types of bottlenecks and applications, our allocation approach can automatically identify resource bottlenecks and allocate the proper amount of resources to each VM such that all the VMs can meet their performance targets if possible. Therefore, the estimation error is a valuable indicator of model accuracy and a valuable guide to the system resource allocator for adaptive allocation. In order to make right resource allocation decisions, accurate workload characteristics and application performance should be captured, estimated, and provide to the resource allocation agent and the allocator. From the experiments, we found that when memory allocation is between 33%-92%, the estimation error is acceptable. However, when memory allocation is less than 33% or larger than 92%, the estimation error increases. This indicates that the model can still be improved. We give the real workloads and estimated workloads in our experiments in Fig. 6.

Due to the disturbances in the collected performance data and the nonlinear nature between the resources consumption and application performance, the estimation accuracy of our model is impacted by the workload strength in experiments, i.e., the resource contention among multiple virtual machines. We provide the accuracy results for response time in httpperf experiments in Fig. 7.

Fig. 6. Accuracy comparison of real traces and estimated data.

Fig. 7. Model accuracy vs. test times under different workload levels (confidence value is 95%).
From Fig. 7 we can see that the accuracy increases when load level increases from 0.5 to 0.9. This is because that in lowly contended condition, resources are idle and data acquisition errors contribute a major component of the performance errors. In highly contended condition (load level = 1.1), the estimation accuracy gradually converge to a stable value. This is because the action of our resource allocator in contended environments.

4.2 Effectiveness

In order to evaluate our algorithms in contending virtualized system, we consider the following four cases:

(1) Case1: No CPU or disk contention

In cases where there is no CPU or disk contention, the physical machine has adequate CPU and disk resources to meet all resource requests, and hence the resources are divided in proportion to the resource requests. For purpose of performance comparison, we repeated the same experiment using the proposed three resource allocation algorithms. The resulting application performances from these algorithms are shown in Fig. 8.

![Fig. 8. Performance of no contention in performance feedback resource allocator.](image)

In the RUBiS testing, we used the default browsing mix workload with 1000 threads emulating 1000 concurrent clients connecting to the bidding server, and the throughput ranges from 50 to about 1500 requests/sec. In default Xen, since it uses a fixed-rate allocation, RUBiS was never able to reach its performance target, and the FTP applications exceeded their targets at some times and missed the targets at other times. Due to the frequent changes in workload behavior, it is very difficult and even impossible to find a fixed allocation ratio for both CPU and disk I/O that will guarantee the performance targets for all the applications. Therefore, in a lightly-loaded environment, the allocation did not provide service differentiation between the applications according to their respective performance targets.

As can be seen, neither approach was able to offer the degree of SLA satisfaction provided by performance feedback resource allocation. For example, in the greedy resource allocation, since the application can utilize the CPU on demand, RUBiS could achieve a throughput much higher than its target at the cost of performance degradation when other applications share the same infrastructure, especially when the contention is intensified.

In the FTP application, we used 50 threads to emulate 50 concurrent clients downloading data at 200KB/sec and measure the total throughput achievable for each ftp application alone. From the results we can see that the ftp application is disk-bound and the
maximum throughput is just above 9MB/sec. However, if the clients request the encrypted data, the test changes to a CPU-bound application and the maximum throughput is around 5MB/sec. The allocation agent identified this change in resource bottleneck automatically and ensured that most of the clients could meet their new throughput targets by allocating the right amount of disk resources to them.

(2) Case 2: CPU contention only

In this scenario, we allocate enough disk resource to meet the requests from the VMs, but not enough CPU resources. In this case, the allocator divides the disk resources in proportion to the requests. However, the applications will receive less CPU resource than requested in order to emulate the CPU contention. In this case, we use priorities for differentiating the allocation between resources.

We run the RUBiS, TPC-W and prime computation simultaneously. The results are listed in Fig. 9 and show that our allocator can support different multi-tier applications during resource contention across the same resources through differentiating priority weights in different applications. In this setup, we simply enhance a higher priority value to the dedicated applications if they are of higher priority as defined in the setting in order to provide service differentiation. The results indicated that CPU contention can adversely impact the application performance.

On server 2, CPU is highly contended. We listed the average completion times of searching prime numbers in the Xen default configuration (with fully fair share of CPUs) and our feedback based allocator configuration in Fig. 10. Our allocator uses 14.06% less time to complete the prime number searching.

From Fig. 10 we can see that when we mix all the applications simultaneously, the performance degrades with respond to the resource contention, but the degradation is acceptable.

(3) Case 3: Disk contention only

In this case, the disk resource is under contention but CPU is not. The allocator follows the same policy for CPU allocation as in case 1, and solves the optimization prob-
Problem to compute the actual disk allocations. We run the ftp server, eMule downloading Oracle operations, and custom file operation benchmark simultaneously. The results are listed in Fig. 11.

In disk contention situations, the performance of ftp server and eMule downloading has a strong relationship with the disk I/O resource allocated. For example, from low to medium levels of competing disk I/O the memory operations per second was constant, but from medium to high levels, memory operations per second decreased substantially. The custom file operation benchmark creates a number of files and performs appends, create, delete, and truncate operations on the pool of files. The benchmark reports transactions per second as the performance metric. In our experiment, we create a 10GB data set for 200000 transactions. We found that transactions per second decreased with increased contending disk I/O. Since it is a meta-data intensive workload and is therefore sensitive to the size of the in-memory file system page cache, transactions per second was greatly influenced by memory and decreased with decreased disk priority.

(a) I/O waiting size of Dom0, Linux01, and Linux02.
(b) I/O waiting size of Linux03, and Windows XP.

Fig. 11. Performance of disk contention only in performance feedback resource allocator.

On server 2, disk I/O is also highly contended. We compared the average I/O wait values in the Xen default configuration and our allocator configuration. Our allocator has 45.59% less I/O wait length to complete the I/O processing.

(4) Case 4: CPU and disk contention

This is the case where both CPU and disk are under contention. In this case, the actual allocations of CPU and disk for both applications will be below the respective requested amounts. We run all the applications simultaneously. The allocator determines the actual allocations by solving the optimization problem. The results are shown in Fig. 12.

In the Oracle application, it performs a fixed number of operations and reports memory operations per second. Obviously, memory operations per second increases with an increase in CPU cap. We created a 4GB table and 40000 database transactions to be processed and the benchmark reports the transactions per second as the performance metric. We found that as memory allocation reduces, memory operations per second remains constant if the requested array hits in the VMs physical memory. However, due to the I/O latencies of paging, memory operations per second reduces super-linearly as memory reduces further.
Since the Oracle database benchmarking is the combination of CPU, memory and I/O devices, we found that when the transactions per second increased with an increase in memory and it was inversely proportional to contending disk io. The author [11] reported that the dependence on \( \text{cap} \) was peculiar as \( \text{cap} \) was increased, TPS increased but beyond a certain point, it started degrading. And the author claimed that this is due to the split-driver architecture of Xen and if the \( \text{cap} \) of the target VM is increased beyond certain value, the available CPU for dom0 reduces with a negative impact on the target VM’s I/O performance. However, we do not observe explicit proofs for this situation.

When running RUBiS, we predefined 4 types of transaction set and we use them to vary the workload with different degree of CPU and IO needs. For each workload, we used 5000 users for all four transaction types, i.e. new customer registration, browse products, orders processing and browse orders. Each user issues a transaction in a closed loop. The timeout for a transaction is set to 60 seconds. We noticed that 50 concurrent users were sufficient to keep the CPU fully utilized. With more users added, the overall latency increases, but the overall transactions per second did not increase much.

In this paper, we used the shopping mix, which simulates a user browsing through a shopping site. The browsing mix stresses the web tier, while the shopping mix exerts more demand on the database tier. In the disk I/O benchmark, reads are done both in forward and backward direction but writes are mostly done to higher block numbers. We can see that reads have a higher IO latency of about 20 ms whereas writes only see a latency of around 5 ms. One reason accounting for this may be that the caching of writes in the storage controller’s cache speedups the write operations. Another reason may be that the write operation uses a bigger I/O size than reads.

In the CPU and disk contention situations, when the concurrent running instances increase by 10%, the performance achieved by each application remains largely the same in the isolated. However, when the concurrent running instances increase by 30% and more, the performance achieved by each application degraded dramatically. For example, the latency is increased in case of resource contention. To identify the responsibility to bursty workload, we change the workload as much as 50% within a few seconds.
4.3 Overhead

In real implementation, resource allocation agent should not add to additional overheads or impede prediction accuracy and runtime performance under various workloads. To identify the suitability of the workload characterization model in a real dynamic environment where the workload may fluctuates frequently, we use a ratio of agent execution time (i.e., $t$) to workload numbers (i.e., $N$) as the overhead metric, i.e.:

Overhead index $= \frac{t}{N}$.

From Fig. 13 we can see that the overhead index is almost constant and does not change much when $N$ increases. In other word, the agent running time is typically proportional to the scale of the workload. This suggests that our agent is adaptive to changes in workload arrival rates and system behaviors, and it can be used by the resource allocator or system administrator to predict workload behavior and application performance online or offline.

5. RELATIONSHIP TO EXISTING WORKS

Virtualization is becoming increasingly popular in large scale data centers and the emerging cloud computing platform because it enables resource sharing and service consolidating while ensuring an almost strict partitioning of physical resources across several applications running on a the same physical platform. However, running such a virtualized system at high efficiency requires sophisticated tuning techniques and is a non-trivial task. Several works have been done for optimized resource allocations.

5.1 Performance Modeling in Virtualized Environments

Since over provisioning and hard partitioning are insufficient and expensive, workload characterization models provide the ability to predict application behavior and per-
formance for a given set of resources and are used for capacity planning and resource allocation. Accurate workload characterization can avoid design inefficiencies and over provisioning of system caused by invisibility into workload details and placement. However, identifying the right resources and services in a specific context in virtualized environments is another challenge in the area of application behavior identification. The literature of current resource allocation has mostly limited to mechanisms of fair sharing and not applicable for better performance such as workload balancing and energy balancing [11-16].

In a virtualized environment, heterogeneous resources, services, and resource requests exist and various among multiple VMs are dynamic, even conflicting. Therefore the above approaches must be modified to adapt to these characteristics because these approaches either do not adequately address all resource dimensions or resource competition or both. To understand the workload behavior and model the resource utilization in a virtualized environment, we consider the influence of resource allocation and resource competition jointly for optimized resource allocation.

Performance prediction is important for resource allocation in both non-virtualized and virtualized environments [17-22]. While the above architecture-specific and performance-counters based models have the potential to provide more precise predictions for a single application, they are difficult to train and hard to use. These application and domain-specific approaches are only suitable for a specific model of the application or the deployment platform and hard to apply to applications running inside multiple VMs running on a shared hardware. Moreover, some of the above techniques is only suitable for the CPU-intensive applications and can only model the CPU resource consumption alone.

Moreover, due to device characteristics and the complexity implicated into the virtualized system in terms of service consolidations, sandboxing, caches contentions, performance modeling is challenging. Some existing commercial virtualization products provide a layer of abstraction for resource management to partition and configure devices as virtual devices needed. Some of them also do automatic load balancing by migrating jobs, data, or VM within or across physical servers. However, configuring systems using these tools is still a difficult task.

In this paper, we look at the overall workload and capacity issues in virtualized system and try to provide more fine grained information about the various components of the workloads. We believe that our simple resource allocation agent is complimentary to the previous work and they would work better with the knowledge gained by our agent. We also provide some insights and guidelines to achieve better performance and utilization through resources allocation.

5.2 Resource Allocation and Scheduling in Contending Virtualized Environments

Recently, control theory has shown its potential in the field of computer systems for resource management and performance control [16, 22-25].

For example, in [22], the proposed multiple-input and multiple-output (MIMO) controller operates on multiple resources (CPU and storage) and uses the sensors and actuators at the virtualization layer and external QoS sensors without requiring any modifications to applications.
However, these methods require parameterized model for virtualized system, which is hard to capture and characterize. In this paper, we use simple and efficient performance metrics such as CPU utilization, memory utilization, and I/O waiting to represent the targeted systems.

5.3 Other Related Work

Traditionally, storage resources are controlled independent of CPU to provide performance guarantees. Dynamic resource allocation is vital for performance guarantees in distributed systems. However, most of the studies focused on allocating resources across multiple nodes rather than in time, because of lack of good isolation mechanisms like virtualization [15, 26, 27, 34]. However, these existing techniques are not directly feasible to allocate resources to applications running in virtualized systems since they cannot provide a way of allocating resources to meet the end-to-end SLAs.

6. CONCLUSION

In presence of the transparency and isolation in virtualized environments, traditional approaches of resource management cannot hold anymore because they need the operating systems to have full knowledge of and full control over the underlying hardware resource. Since the VMM usually does not know what the VMs are doing and the VM are unaware of the activities of the underlying hardware, it is hard for VMM to implement efficient and effective resource management decisions while guarantying the SLA requirements. Moreover, since the VMM are unaware of the activities of the upper level applications inside various guest VMs, it is hard to make resource allocation decisions without sufficient workload characteristics.

In this paper we proposed a resource allocation agent and stochastic model for resource allocation. We also presented a workload characterization model and techniques for resource allocation over a set of VMs. We used parameters independent of specific undertaking hardware for modeling and statistics. The proposed algorithms respond to resource allocation by using closed-loop policies to set a safe performance level and it considers the coordinated optimization of different SLA requirements together. It also attempts to address hotspots problem among multiple VMs through migrations.

The proposed approach is implemented as a prototype in Xen-based virtual machine environments and the experiments show that it is accurate among multiple VM instances and across a range of applications. The results show that the proposed scheme can achieve a higher SLA satisfaction rate as 97.1%, a lower load imbalance index as 18.7%. The performance feedback based allocator uses 14.06% less CPU time for CPU-intensive applications and reduces 45.59% I/O wait time in disk contention environments. It is clear that profound changes in system performance, robustness, reliability and scalability can be achieved by performance feedback resource allocation algorithm in contending environments. The results suggest the suitability of our approach in virtualized environments.

In this paper we do not consider additional resources, such as cache allocation and contention. The refining research may further increase the prediction accuracy and allocation efficiency in the future work.
REFERENCES


Cong-Feng Jiang (蔣從鋒) received his Ph.D. degree from the Engineering Computation and Simulation Institute (ECSI), Huazhong University of Science and Technology (HUST), Wuhan, China, in 2007. He is now an Associate Professor in School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. He is with the Grid and Services Computing Lab. His current research interests include virtualization, grid computing, and power aware computing system.

Jiang Wan (萬健) received his Ph.D. degree in Computer Science from Zhejiang University, China, in 1996. He is now a Professor and Dean of School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. His research interest includes virtualization, grid computing, services computing, and wireless sensor networks.

Xiang-Hua Xu (徐向華) received his Ph.D. degree in Computer Science from Zhejiang University in 2005. He is now a Professor in School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. His research interest includes virtualization, grid computing, services computing, and wireless sensor networks.
Ji-Lin Zhang (張紀林) received his Ph.D. degree from the School of Computer Science and Technology, Beijing University of Science and Technology, Beijing, China, in 2009. He is working in the Grid and Service Computing Lab, Hangzhou Dianzi University, Hangzhou, China. His current research interests include virtualization, parallel computing, and distributed computing system.

Xin-Dong You (遊新冬) received her Ph.D. degree from the School of Computer Science, Northeastern University, Shenyang, China, in 2007. She is working in the Grid and Service Computing Lab, Hangzhou Dianzi University, Hangzhou, China. Her current research interests include virtualization and grid computing.